

## **REMARKS**

Claims 1-15 were pending in the present application. Claims 1, 4, 6, 9, 11, and 14 have been amended. Accordingly, claims 1-15 remain pending in the application.

Claims 1-15 stand rejected under 35 U.S.C. 112, 2<sup>nd</sup> paragraph, as being indefinite for failing to particularly point out and distinctly claim the subject matter which the Applicant regards as the invention. The Applicant has amended claims 1, 4, 6, 9, 11, and 14 to overcome the rejection. Specifically, as shown above, the Applicant has made various minor amendments to the claims to correct typographic errors and antecedent issues.

Claims 1-15 stand rejected under 35 U.S.C. 103(a) as being unpatentable over Inakoshi (U.S. Patent Number 5,933,604). The Applicant respectfully traverses this rejection.

Claims 3-4, 8-9, 13-14 stand rejected under 35 U.S.C. 103(a) as being unpatentable over Inakoshi in view of Conger “Windows API Bible 1992” (hereinafter ‘Conger’). Applicant respectfully traverses this rejection.

Claims 5, 10 and 15 stand rejected under 35 U.S.C. 103(a) as being unpatentable over Inakoshi in view of Arthur Dumas “Programming WinSock 1995” (hereinafter ‘Dumas’). Applicant respectfully traverses this rejection.

The applicant discloses at page 5, line 23 through page 6, line 26

“If the process 12(n')(m') is waiting for information from the process 12(n)(m), which may occur after it issues a processing request to the process 12(n)(m), it (that is, process 12(n')(m')) will perform a “spin-wait” loop in which it periodically tests the flag 16(n')(m') while performing the spin-wait loop, the process 12(n')(m') will need to delay performing other processing operations until the process 12(n)(m) has loaded the information in its memory portion 13(n')(m') and set the flag

16(n')(m'). ... To reduce the amount of processing time that may be wasted by the process 12(n')(m') executing the spin loop, the computer 10 also includes a spin daemon 17 which, after receiving a flag monitor request therefor from the process 12(n')(m'), can monitor the condition of the flag 16(n')(m') for the process 12(n')(m'). The process 12(n')(m'), after issuing a flag monitor request to the spin daemon 17, can thereafter de-schedule itself, by removing its identification from the task list 15, and provide a notification to the operating-system 14 giving up any remaining portion of its current time slot. ... After receiving the flag monitor request from the process 12(n')(m'), the spin daemon 17 will add the flag monitor request to a spin list 18 which it maintains. The spin list 18 identifies the flags in memory whose condition it is to monitor. Accordingly, after the spin daemon 17 adds the flag monitor request to the spin list 18, it will thereafter monitor the condition of the flag 16(n')(m'). When the process 12(n)(m) sets the flag, indicating that has provided the information required by the process 12(n')(m'), the spin daemon 17 will enable the process 12(n')(m') to, in turn, enable the operating system 14 to re-schedule the process 12(n')(m') by reloading its identification on the operating-system task list 16." (Emphasis added)

Accordingly, the Applicant's claim 1 recites

"a system for controlling co-scheduling of processes in a computer comprising at least one process and a spin daemon, the process being configured to, when it is waiting for a flag to change condition, transmit a flag monitor request to the spin daemon and de-schedule itself, the spin daemon being configured to, after receiving the flag monitor request monitor the flag and, after the flag changes condition, enable the at least one process to be re-scheduled for execution by the computer." (Emphasis added)

Applicant's claim 2 recites

"said spin daemon is configured to monitor a plurality of flags, each in response to a flag monitor request, the spin daemon maintaining a list identifying those flags it is to monitor, the spin daemon being further configured to, when it receives a flag monitor request, add an identification of a flag associated with the request to the list." (Emphasis added)

The Examiner asserts that Inakoshi teaches a system that includes the features recited in Applicant's claim 1. Specifically, the Examiner asserts that the at least one process as recited in the Applicant's claim 1 is disclosed in Inakoshi at col. 4, lines 26-28 "the monitor request process **from a user.**" The Examiner also asserts that Inakoshi discloses at col. col. 4 lines 31-37, wherein the at least one process transmits a flag monitor request (monitor the state of the resource request process **from the user**) to the

spin daemon, and after the flag changes condition, enable the process to be rescheduled for execution by the computer. Applicant respectfully disagrees with the Examiner's assertions and characterization of Inakoshi. (Emphasis added)

More particularly, Inakoshi is directed toward a network resource monitoring system in which **users** (i.e. persons), send requests to monitor an information resource on a communication network to a management system. [See abstract]

At col. 4, lines 26-28, Inakoshi discloses "The monitoring unit 1 monitors the state of the resource 6 on the communication network 5 based on **a request from a user**." (Emphasis added)

Inakoshi further discloses at col. 4 lines 31-37 "The resource 6 is, for example, image information and/or text information, such as a home page on the Internet, that is created and transmitted **by a user**. Which resource on the communication network 5 is the target of monitoring is specified at the time of the user request. The **monitoring request from the user** is sent, for example, from the management unit 4 to the monitoring unit 1." (Emphasis added)

It appears the Examiner is asserting that **a user** (i.e. a person) making a request is analogous to a process (e.g., a software routine) running on a computer. The Applicant submits that a person submitting a request **is not the same** as a software process running on a computer.

Furthermore, the Examiner acknowledges that Inakoshi does not explicitly teach "the process" descheduling itself. The Examiner has asserted that the user does not have to access the network resource and thus, it would have been obvious that the process from the user has been descheduled from accessing the resource by itself.

The Applicant submits that it is well known in the art that the phrase a process configured to "deschedule itself" refers to a process determining for itself that it will not

execute any further and then causes itself to not be executed. Specifically, in this context, a process might deschedule itself so that another process that is not waiting for a flag may execute instead. The Applicant can find no reference in Inakoshi to **the user** doing anything that can be construed as descheduling itself. Furthermore, the Applicant fails to find any reference in Inakoshi to “enable the process to be rescheduled for execution by the computer.”

From the foregoing, Applicant submits that Inakoshi **does not teach or suggest** “a system for controlling co-scheduling of processes in a computer comprising at least one process and a spin daemon, the process being configured to, when it is waiting for a flag to change condition, transmit a flag monitor request to the spin daemon and de-schedule itself,” as recited in Applicant’s claim 1. Further, Inakoshi **does not teach or suggest** “after the flag changes condition, enable the at least one process to be re-scheduled for execution by the computer” as recited in Applicant’s claim 1.

In addition, Inakoshi teaches at col. 9, lines 35-40 “In addition, the communication network 49 is also connected to the resources 50-1, 50-2 and 50-3 that are identified by the URL. The number of resources that are connected is arbitrary. Hereafter, a resource such as this will simply be referred to as "URL". (Emphasis added)

Thus, Inakoshi **does not teach or suggest** “said spin daemon is configured to monitor a plurality of flags, each in response to a flag monitor request, the spin daemon maintaining a list identifying those flags it is to monitor, the spin daemon being further configured to, when it receives a flag monitor request, add an identification of a flag associated with the request to the list,” as recited in claim 2.

Conger is directed to the Windows operating systems and memory management. Conger discloses at page 612 lines 10-11 “The 8086 and 80486 chips access memory using two 16-bit values. These values are called the “segment” and the “offset.” The Applicant can find **no reference whatsoever** in Conger to using a memory segment which contain monitor flags.

Dumas is directed to using the Windows Sockets Application Programming Interface. Dumas does not teach or disclose the combination of features recited in Applicant's claim 1, 6 or 11.

The Applicant submits that claim 1, along with its dependent claims 2 – 5, are believed to patentably distinguish over Inakoshi, Inakoshi in view of Conger and Inakoshi in view of Dumas for the reasons given above.

Claims 6 and 11 recite features that are similar to the features recited in claim 1. As such, claims 6 and 11, along with their respective dependent claims 7 – 10 and 12 – 15, are also believed to patentably distinguish over Inakoshi, Inakoshi in view of Conger and Inakoshi in view of Dumas for at least the same reasons.

## CONCLUSION

Applicant submits the application is in condition for allowance, and an early notice to that effect is requested.

If any fees are due, the Commissioner is authorized to charge said fees to Meyertons, Hood, Kivlin, Kowert, & Goetzel, P.C. Deposit Account No. 501505/5181-93900/BNK.

Respectfully submitted,



Stephen J. Curran  
Reg. No. 50,664  
AGENT FOR APPLICANT(S)

Meyertons, Hood, Kivlin, Kowert, & Goetzel, P.C.  
P.O. Box 398  
Austin, TX 78767-0398  
Phone: (512) 853-8800

Date: 6/10/04